Quality-Aware Resource Management for Wireless Sensor Networks *

Roland Gémesi, Nirvana Meratnia, and Paul Havinga

University of Twente, Department of Computer Science,
P.O.Box 217, 7500 AE, Enschede, The Netherlands
{r.gemesi,n.meratnia,p.j.m.havinga}@ewi.utwente.nl

Abstract. Due to the distributed and resource constrained nature of wireless sensor networks, their design proves to be difficult. We present a resource management framework, which integrates a data-centric lightweight operating system with a publish/subscribe middleware. In this framework, the main system abstraction is data for both local and networked processing. The resultant system software is extended with a quality-aware adaptation mechanism, which configures system timeliness according to the actual application requirements. A feedback-based control mechanism is used to iteratively tune the resultant data granularity in order to fit user requirements. Our design is evaluated by simulations and the concepts were also implemented in our sensor network testbed.

1 Introduction

Wireless Sensor Networks represent the first step towards ubiquitous computing [1]. They are massively distributed systems built of small embedded devices with computing and wireless communication capabilities and provide data-collection or possibly real-time control functionality by means of collaboration. Typically, a number of sensor nodes are scattered over the target area to monitor and provide information about requested phenomena.

The considered distributed platform differs from traditional computing systems in numerous aspects. Sensor nodes are usually small, battery powered, and possess extremely limited hardware resources, such as CPU and storage capacity. Due to the finite battery supply, the utilised mechanisms have to be as energy-efficient as possible to maximize the lifetime of the system. Despite these constraints, complex tasks can be performed by means of collaboration. These embedded devices use self-organised wireless networking to interact with each other; however, their communication is very limited as well. Since nodes may arbitrarily arrive or leave the network, the available resources can dynamically change during the operation. Moreover, both nodes and their communication are characterised by uncertainty.

Designing distributed applications in a harsh environment having the above-mentioned characteristics proves to be difficult. The management of distributed,

* This work was funded by STW under the Featherlight Project number TES-6388.

scarce and dynamic system resources is too complicated to be performed by each application, consequently mechanisms are needed to automate common distributed system functionalities. Moreover, the possibly immense number of nodes might produce a vast amount of information, usually much more than required. The quality and granularity of the acquired output should be managed according to the actual user/application requirements. Thus, resource management should also be capable of adapting service qualities, consequently managing what resources are spent for.

We introduce a system software framework that integrates a data-centric real-time operating system with a publish/subscribe middleware. This results in a resource management framework, which presents similar system abstractions for local and networked application development. We extend the resultant resource management framework with quality-awareness, which supports dynamic quality-aware reconfiguration of distributed mechanisms based on the application requirements. The provided framework is evaluated through simulations and the mechanisms are also implemented in our sensor network testbed.

The rest of this paper is organized as follows. Section 2 introduces the problems that motivate our work in more details. The middleware that extends the local abstractions of our local operating system is proposed in Section 3. Section 4 describes our feedback-based quality-management mechanism, followed by evaluation of the proposed framework in Section 5. Section 6 overviews related work and finally Section 7 concludes the paper.

2 Motivation

Collaborative applications involve several entities and utilise several resources concurrently in order to fulfil a common goal. The management and coordination of such distributed entities make the development very complex. Moreover, dealing with the dynamic and uncertain nature of the underlying system poses additional burden on the application developer.

The issue of handling system complexity is already present in traditional distributed systems, in which the objective of *System Software* is to support managing the system, consequently easing application development [12]. System software presents programming interfaces to perform standard operations, thus prevents designers to handle all the underlying issues of the actual system. This results not only in a simpler and generic development framework, but also helps avoiding reimplementing common features in each of the applications. Traditionally, the system software of distributed systems was divided into three parts: (i) *Operating System* (OS), which hides low-level hardware concepts by providing APIs for standard concerns, such as starting or stopping processes or allocating memory (ii) *Communication Protocol Stack*, which decomposes network communication tasks into a set of standardized layers (iii) *Middleware* that aims at hiding the networked nature of the underlying system and providing high-level, often application-dependent, programming primitives for distributed applications.

Providing generic system primitives calls for abstractions that shield developers from the underlying complexities of the distributed system. These high-level concepts should be useful for development purposes, however should not contain any details regarding the specific low-level system implementation. Selecting suitable abstractions for sensor networks is especially challenging. The complex distributed nature of the system makes it difficult to provide simple interfaces that can be maintained with a wide range of resources. For instance, the offered programming primitives should be independent of the number of available nodes or the topology they are actually organised in.

On top of that, the resources to manage can change dynamically. As new nodes enter the network, their offered services should get involved in the collaboration, although, when nodes leave or fail during operation, the system should tolerate the lack of their effort. The dynamic changes occurring in the actual network should not be realised by the applications that rely on high-level system primitives. System support should resolve such changes by reacting and adapting to changing resources.

As severe resource changes occur, some system services might not be maintained without degradation. However, in sensory environments *quality-resource tradeoffs* are usual between the accuracy of their acquired readings and the resource consumption. The system might operate at several quality levels, however the availability of resources and the minimal quality requirement specified by the application decide on suitable choices.

Resource management of sensor networks should consider such tradeoffs as tunable *knobs* of the system. Adaptively influencing quality metrics and consequently regulating resource demands has considerable benefits. It increases the resource-tolerance, because in case of insufficient resources the system might keep on operating with decreased quality levels. On the other hand, even if resources allow high granularity, avoiding acquiring unnecessary details saves energy. Thus, resource management should not only satisfy mechanisms with dynamic resource availabilities, but also manage quality-resource tradeoffs to regulate resource demands.

In this article, we introduce the design and evaluation of a resource management framework that provides system primitives suiting dynamic systems. The aim of our design is to set up a framework, in which both local and distributed applications rely on similar system primitives. Moreover, the proposed system software is capable of adapting qualities according to the available resources and user requirements, thus actively influencing the timeliness of system mechanisms.

3 Data-Centric System Software for Sensor Networks

A major issue in designing system software is to find the suitable system abstractions that are practical for common application needs and are not related to the actual resources. Since sensor networks are built of processing elements that might arrive and leave the network arbitrarily, processing models that explicitly describe the actual processing flow are not practical.

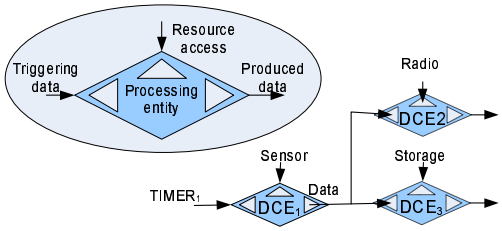

Fig. 1. Data-Centric Entities in AmbientRT

Our framework follows the data-centric concept [5]. In a *data-centric* system, actions are triggered by the data that is available in the network. Thus it focuses on the data being produced and consumed, but does not explicitly define the actions to be taken with produced data. That is the role of the system to pass the data to a suitable processing entity and activate it.

In the following, we show both the operating system and middleware levels of our data-centric system software.

3.1 AmbientRT

AmbientRT [8] is an embedded operating system developed and successfully applied by our research group. It was designed to fit the extremely limited memory and computing capacity of sensor nodes, while being able to provide real-time guarantees and facilitating energy-efficient operation.

AmbientRT is a data-centric operating system, in which the main system abstraction, called an *event*, can range from a system event to data measured from an environmental phenomena. The system is built of software components called *Data Centric Entities (DCE)*, which perform the processing tasks. As illustrated in Figure 1, the execution of DCEs are triggered by the availability of a data. During their execution, they usually require system resources and produce output data that can trigger other entities.

In the example illustrated in Figure 1, entity DCE_1 is activated by the periodic event *timer*, and perform a sensor *measurement*, presenting output data as a result. This periodically produced data activates two other entities: DCE_2 and DCE_3 representing storage and radio transmission functionality.

The central element of AmbientRT is the *Data Centric Scheduler*, which keeps track of all entities and has two main functionalities: to control which entities should be activated and to manage the data flow between the entities. The kernel uses a scheduling method called *Earliest Deadline First with Inheritance* [10] that enforces mutual exclusion of shared resources without the need of semaphores. This scheduling mechanism enables entities to meet real-time requirements.

The data-centric architecture of AmbientRT enables dynamic runtime reconfiguration, which makes it especially suitable for dynamic environments. As new

entities appear and others are removed, the execution flow reconfigures automatically. AmbientRT makes the development of data-centric applications simple, enabling the developer to focus on evolving data instead of tracking dynamically available entities.

AmbientRT is a lightweight operating system facilitating a data-driven model for application development. It suits systems with limited and dynamic resources, but focuses only on the local aspects of system management. Since we consider distributed systems, it is still difficult to implement networked applications by using only local system support.

3.2 Publish/Subscribe Middleware for AmbientRT

To facilitate the development of networked applications, a network-wide resource management framework is required. We propose a middleware that extends the concepts of AmbientRT and provides efficient support for developing distributed applications.

To suit the data-centric philosophy, we apply a *publish/subscribe middleware*. In a publish/subscribe middleware, some nodes publish data that others are subscribed for. These abstractions are independent of the actual network topology and of available resources; it is focused on the data, therefore it is suitable to extend our operating system. The role of the middleware is to offer simple application interfaces and to manage disseminating data to the subscribed nodes.

The communication of the proposed middleware relies on the services offered by LMAC, our energy-efficient medium access control (MAC) protocol [14]. LMAC uses *Time Division Multiple Access* (TDMA) to share the communication channel among the participants. It divides time into slots and sets up schedules that determine when the nodes are allowed to transmit. LMAC uses a distributed algorithm that relies only on local information to create the schedules. LMAC is especially efficient in resource constrained environments, because nodes that do not communicate in a time slot might turn off their radios, consequently saving significant amount of energy.

LMAC also includes a basic routing functionality, which enables nodes to send data to the sink node, consequently supporting data collection in sensor networks. Each node chooses one of its neighbours that is closer to the sink node, and maintains it as a parent in the data collection tree. Our middleware relies on this data collection tree, thus no energy is wasted to set up an additional routing hierarchy.

With this basic routing support, we aimed at building a multicast routing scheme to disseminate data to the subscribed places. Our routing scheme is similar to the Protocol Independent Multicast (PIM) [4] protocol. Similarly to PIM, subscribers send requests to a rendezvous point setting routing states along the path. A published message is first sent to the rendezvous point, which subsequently routes it to all subscribers. However, PIM was proposed for wide area networks, and our mechanism contains additional optimizations to save as much energy as possible.

ST_i : subscription table of node i
PST_i^j : proxied subscripiton table of node i for neighbour node j
$uplink_i$: uplink of node i
$downlink_i^j$: downlink of node i to neighbour j
$inlink$: the link on which event e arrived from

$Procedure_1$: node i subscribes for event e:
    if $((e \notin ST_i) \wedge (\neg\exists j : e \in PST_i^j))$ send e on $uplink_i$;
    add e to ST_i;
End;

$Procedure_2$: node i publishes event e:
    if $(e \in ST_i)$ dispatch to local operating system;
    send e on $uplink_i$;
    $\forall j$: if $(e \in PST_i^j)$ send e on $downlink_i^j$;
End;

$Procedure_3$: Subscribing network packet arrives:
    if $((e \notin ST_i) \wedge (\neg\exists j : e \in PST_i^j))$ send e on $uplink_i$
    add e to PST_i^{inlink};
End;

$Procedure_4$: Publishing network packet arrives:
    if $(e \in ST_i)$ dispatch to local operating system.
    $\forall j$: if $((e \in PST_i^j) \wedge (j \neq inlink))$ send e on $downlink_i^j$;
    if $(inlink \neq uplink_i)$ send e on $uplink_i$
End;

Fig. 2. Pseudocode of Event Dissemination

The nodes of the proposed publish/subscribe middleware perform as follows. When a node subscribes for a data type, it sends a *subscription message* to the root node. This message registers the interest both at the root as well as at each intermediate hops, so the delivery paths from the root node towards each interested parties are formed. When an arbitrary node wants to publish a data, it sends the data to the root node first, which disseminates it along all the interested links of the tree. The pseudocode of the message dispatching algorithm run by the nodes is shown in Figure 2. The actions of the root node differ slightly, because it cannot send messages further in the uplink direction.

Each node i in the network is a participant of the data delivery tree, consequently each node maintains links both to its parent ($uplink_i$) and to all its descendants ($downlink_i^j$). Since the tree is set up by LMAC, the middleware assumes that this information is available. The middleware registers subscriptions in order to maintain the dissemination tree. Each node i stores a *subscription table* for maintaining its own interest (ST_i), and also *proxied subscription tables* (PST_i^j) for the interests of all the descendants.

When a node subscribes to a new data type, an entry is added to its local subscription table. Then, if the data is neither present among the local nor the

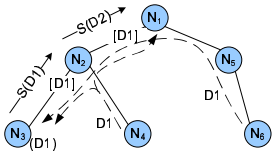

Fig. 3. Example Data Dissemination in the Proposed Middleware

proxied subscriptions, the subscription message is sent to the parent. If the data is already present in any of the subscription tables, an interest has already been registered at the parent, thus it is not necessary to do it again. Similarly, as a subscription message arrives, the packet is sent to the parent only if it has not been sent before. The incoming interest is stored in the proxied subscription table of the particular incoming link.

When data is published, the middleware disseminates it to all the subscribed parties. If the actual node itself is subscribed, the message has to be dispatched to the local operating system first. Afterwards, the publishing message is disseminated to the parent node, and to each of the interested descendants. When a publishing network packet arrives, it is not necessary to disseminate the message on the incoming link. Thus, if the publishing message arrived from a descendant, it is sent to the parent and also to each interested downlinks, except the one it arrived from. When a publication arrives from the parent, it is disseminated only to the interested descendants.

An example is illustrated in Figure 3. Nodes ($N_1 \dots N_7$) form a tree rooted at N_1. When N_3 subscribes for data type D_1, its subscription message (S_{D_1}) is sent to N_2 first. N_2 registers that the link towards N_3 is interested in D_1, then it forwards S_{D_1} to the root (N_1), which registers the interest of N_2. When N_6 publishes D_1, it is first delivered to the root through N_5, then it is sent through the previously registered route to N_3. However, when N_4 publishes the data, N_3 gets the message directly from N_2 even before N_1 receives it, thus making a shortcut in the network.

Although not shown in the pseudocode, the routing mechanism can recover from errors, because the subscription tables hold enough redundancy to avoid loosing routing information. When a branch of the tree is broken, the root of the subtree knows all subscription information of the nodes below it. Thus, as it is reattached to an arbitrary parent, it can send its subscription tables in one step, and the system is just ready to operate again.

Since LMAC is a schedule-based protocol, nodes that are willing to transmit have to wait first until their time slot arrives. However, the delay of transmissions is quite deterministic. It was shown previously, that when time slots are randomly chosen amongst the available slots, the average transmission delay is a half time frame. Because our routing uses the shortest possible path to connect the sink to the receivers, the resultant dissemination delay is the smallest possible, dependent on the depth of the tree.

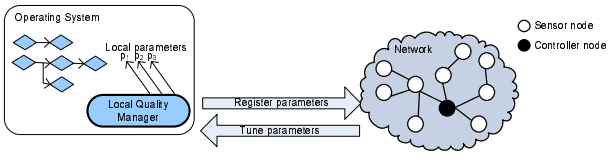

Fig. 4. Quality Management in a Data-Centric System

The introduced mechanism runs at each node of the network, forming a collaborative middleware service. Towards the operating system, it provides interfaces to publish or subscribe data. Consequently, the data-centric entities of the local operating system can be easily extended to publish their output events network-wide. Other nodes, that are subscribed, automatically get the data delivered. The dissemination of data and the management of the network are not a burden on the application developer any more.

The mentioned middleware extends the data-centric concepts of AmbientRT for networked applications. The events of AmbientRT can be easily exported to the network, thus making local and networked processing almost identical from the development point of view. The proposed middleware relies on the communication services offered by LMAC, thus efficiently utilising already existing communication infrastructure. However the solution is centralized at the root node, such topology is reasonable in systems that perform data collection.

4 Quality-aware adaptation

The resource management of sensor networks should be capable of managing the hosted mechanisms by controlling their resultant quality levels, consequently regulating their resource demands. With the support of the introduced data-centric architecture, we propose a quality management scheme, which adapts network services according to application/network/user requirements. In this section, the applied architecture and our feedback-based quality management mechanism are introduced.

4.1 Quality management architecture

The mechanisms capable of operating with several quality levels usually possess parameters that influence the granularity of their output. Unfortunately, these parameters are often kept as fixed internal values, not supplying any interfaces to change them. To facilitate the system managed tuning of quality levels, we made the introduced system software capable of accessing and adjusting these *control knobs* accordingly.

The idea is depicted in Figure 4. We let the processing entities of AmbientRT define their parameters, which manipulate the quality of their resultant data. The quality tuning changes only the accuracy of the provided information but

not the data type itself. Entities offer these knobs to the local operating system, where a *local quality manager* is in charge for setting them to suit the actual requirements.

We utilised the proposed publish/subscribe middleware to facilitate network-wide quality management. The network of local quality managers utilise data-centric middleware services to enable remote quality changing. Thus, they subscribe for control messages and rely on the middleware to get control messages relevant to local knobs delivered. When such control occurs, the middleware is in charge for delivering the control messages to each tunable party. As a result, all members of the given distributed service are configured accordingly.

For instance, the distributed service providing periodic temperature measurements might offer knobs to influence the sampling frequency or the resolution of the measurements. The processing entities thus offer their knobs for the operating system, which subscribes for them at the middleware. During the operation of the system, such control messages can speed up or slow down measurements.

The resultant quality management scheme is thus capable of reconfiguring collaborative services, since it delivers data to all its members. By relying on the dissemination service offered by the middleware, the required implementation effort remains quite small.

4.2 Feedback-based quality control

Even though the knobs can be controlled network-wide, choosing an adequate configuration is challenging.

Control messages might be sent by any of the network participants, but we assume that a particular entity exists for quality management. This entity might be the base station, the network maintainer or the user of the application. The manager has to evaluate the actual conditions and make decision about the suitable control parameters accordingly. The choice is determined by the amount of available resources and the application requirements.

Using a centralised quality manager is a limiting factor, because in many cases local control needs to be executed. For instance, nodes around areas of interest need to perform more accurate measurements than nodes that do not observe any interesting data. Such local control can be performed by forming multiple smaller groups, and executing the control mechanism within the group.

Applications express their interest by *Quality of Interest* (QoI) specifications. If the terms of QoI specify high-level properties, such as the confidence of pattern detection algorithm, determining the required parameters requires additional models. Thus, the terms in which the QoI is expressed might require application-dependent knowledge to facilitate finding the parameters resulting in the required quality.

Such models are usually not available, but the QoI requirements can also be expressed in more generic terms. We focus on the case, when QoI is given as statistical requirements over the provided data. For instance, QoI might specify the number of required samples or the standard deviation of the result. Such

metrics can be evaluated independently of the actual application, therefore the parameter adjustment results in a more general framework.

Since such statistical properties can be evaluated based on purely resultant data, we apply a feedback-based approach to continuously adapt the system to meet the requirements. As shown in Figure 5, the quality of the result is continuously evaluated and compared to the expectations. If the difference makes it reasonable, parameters of the mechanism are changed.

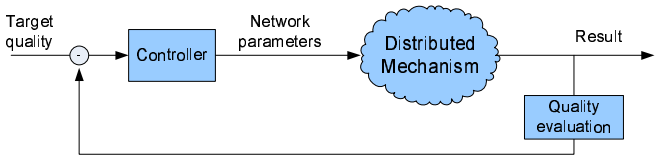

Fig. 5. Closed Loop Quality Control

The design of such a closed loop control is well supported by the design framework of digital control engineering. The basic task is to find the suitable controller for the given real world system to achieve an expected output. However, when choosing the controller the behaviour of the controllable system has to be considered. Whether the actual statistical QoI involves delays or transient events, the controller has to fit the proportional, integral or derivative nature of the controlled system.

Thus, our proposed quality-management scheme is based on the continuous analysis of results and applies closed-loop feedback control to satisfy requirements. Since application-dependent models might support abstract QoI adjustment, in case of purely statistical requirements such models are not required.

5 Evaluation

We have investigated the efficiency of the proposed schemes by simulations as well as by real experiments. This section introduces the evaluation of the middleware and the quality adjustment scheme.

5.1 Overhead of data dissemination

First we evaluate the efficiency of the proposed publish/subscribe middleware. Since the routing topology of LMAC was used, our focus is to investigate how efficiently it is used and not to examine the tree itself.

We implemented the proposed publish/subscribed scheme in the OMNeT++ discrete event simulator [11]. For simulations, we applied a random topology of 40 nodes scattered uniformly in a 600x600m area. The communication range was given 125m.

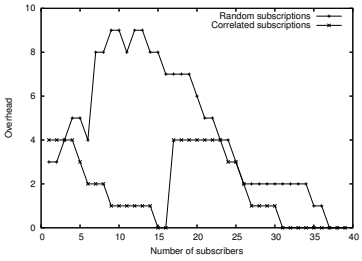

Fig. 6. Overhead of the Data Delivery Tree

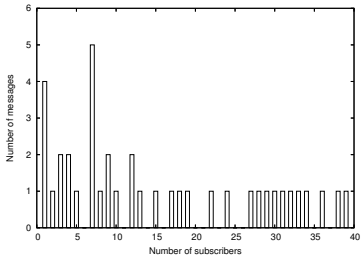

Fig. 7. Messages Needed to Register New Subscriptions

First, we investigated the overhead of the dissemination tree. We define *overhead* as the number of nodes participating in the data dissemination without being among the subscribers. We have changed the number of subscribers from 1 to 40 and recorded the number of such nodes. We evaluated both cases when the subscribers were randomly selected and geographically nearby. The results are shown in Figure 6.

When the subscribers were randomly selected, the overhead grows for a while and then decreases to zero. It grows as long as new branches became involved in the delivery tree, possibly involving nodes to forward the message without actually being interested in it. As the tree becomes more and more saturated with subscribers, the chance that such a purely forwarding node becomes a subscriber grows. Consequently, the overhead decreases and when all the nodes are subscribed for the data, the overhead reaches zero.

The overhead has a different trend when the subscribers are geographically nearby. Nearby nodes often have the same node as their parent, thus the overhead does not grow as new nodes subscribe. Moreover, if the chosen node is the parent of a previous subscriber, the overhead just decreases. Consequently, the overhead gradually decreases, except for the case when a completely new branch of the tree becomes involved in the dissemination. It can be seen from Figure 6 that the overhead reaches zero when 15 subscribers are present. In this case, all members of a tree branch are subscribed, however after this point a completely new branch has to be involved again.

Another interesting metric is the number of messages needed for a new node to join the data delivery tree. The subscription is sent upwards in the tree until it reaches either the root or a node that is already subscribed. Figure 7 shows how many times the subscription messages has to be transmitted. As expected, it is always less than the depth of the tree, which is actually 6. It can be seen, that except the two peaks at second 1 and 7, the number of required messages is quite small. The reason that peaks occur is that as nodes in a new branch subscribe, the subscription message should travel several hops upwards in the tree to meet an already registered node.

Our evaluation focused on how efficient our data delivery service is over a given routing tree. We have evaluated the overhead, and have seen that it

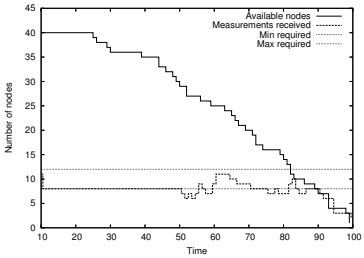

Fig. 8. Number of Received Samples in a Time Interval

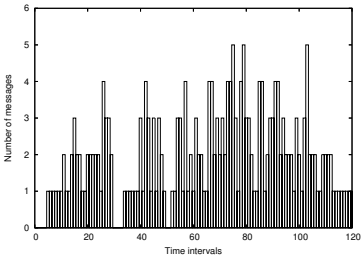

Fig. 9. Experimental Results of Received Samples

depends on the spatial distribution of subscribers. The dissemination is usually more efficient if the subscribers are nearby. We have also seen that the number of messages required for subscriptions usually remains quite small, because it has to reach only the closest actually subscribed node.

5.2 Quality-aware adaptive control

We evaluate the feedback-based quality control mechanism through a sample. Assume a typical sensor network to collect measurements periodically over a target area. The QoI specification describes how many of the nodes need to provide measurements. It specifies minimum and maximum thresholds for the number of samples received by the sink node. Since the number of available sensors might be very large, it is sufficient to sample only a subset of nodes and turn the rest off or to power-saving mode. The sink evaluates the received samples in specified time intervals and controls the system accordingly.

To influence the number of nodes participating in the sampling process, we use probabilistic sampling with a p probability value as a parameter. Each node receiving p decides to participate the measurements with probability p and is turned off otherwise.

The sink node is in charge of determining p according to the received number of samples. Since the controller entity is also a resource constrained unit, we decided to keep the controller mechanism as simple as possible. Changing the applied p probability results in immediate changing of the number of collected samples and does not contain dynamic effects. Consequently, the applied controller adjusts p proportionally to the ratio of required and desired number of samples.

First, we evaluated the control mechanism by simulation, in which the previously described setup of 40 nodes was used. The root node collects the measured data, and performs the control in every second if it is necessary. The QoI requirement was chosen to require 8 to 12 samples per interval. The simulation starts with a stationary environment, then after second 25 nodes die uniformly.

The number of actually received samples and the number of available nodes can be seen in Figure 8. First, it can be seen that despite the large number

of nodes, only the required number of samples was produced. Second, notice that as the number of available nodes decreases, the number of received samples mostly remains within the specified interval. When the specification is not met (for instance at second 50), the network is immediately re-adjusted so that the target QoI is met again. Finally, after second 88 the requirements cannot be met any more.

Besides, we have also implemented this quality-aware control mechanism in our sensor network testbed and evaluated it through real experiments. The target number of samples was set to be either 2 or 3 in each 4 seconds time intervals. We used 10 sensors that are gradually deployed in the first half of the time, then started to turn them off. The number of samples received by the sink node are shown in Figure 9. It can be seen that the number of received samples has never reached 10, thus the system did not activate some of the sensors in order to save their battery capacity. After deploying all the nodes, we started turning off those nodes that were actually sampling. No degradation can be seen in the second half of the time, thus the system reacted to changes and reconfigured well in order to maintain the specifications.

Although the number of acquired samples did not grow more than 5, unattractive fluctuations can be observed. The reason is that as some samples were missing due to transmission errors, the system thought nodes became unavailable and proportionately increased the probability. Since the actual node was available at the following sampling, the resultant amount of acquired information was increased unnecessarily. Thus, the system overreacted communication errors and resulted in transients. However these transients might be suppressed by filtering techniques, which would result in increased responsiveness and longer convergence time.

As we have shown, adjusting sampling parameters based on the QoI requirements is efficient to avoid unnecessary actions. Resource management controls system qualities to meet and maintain the target quality level even during resource changes. With the support of the proposed publish/subscribe middleware; controlling of distributed control knobs is simple. However, the experimental results show that even if the controllable quantity does not produce dynamic behaviour, communication uncertainties might call for more complex controllers.

6 Related work

The massively distributed, dynamic and uncertain environment poses several new concerns to system software [12]. To respond to these concerns, several operating system platforms have been proposed for extremely resource constrained systems. TinyOS [7] is one of the first and best known among them. Its component-based development model is easy to use, however it is not capable of providing real-time execution guarantees and the dynamic reconfiguration of system components is not easy. In contrast, AmbientRT [8] is a data-centric operating system, enabling real-time execution guarantees and supporting simple runtime reconfiguration.

In addition to operating systems, numerous sensor network middleware platforms have been evolved to provide high level distributed services. Cougar[2] and SINA [13] are two middleware platform examples, both having database-like query processing support, which is usually required by typical sensor network applications. Unfortunately, these platforms are not able to guarantee quality of service requirements.

QoS support for Wireless Sensor Networks can be categories into network, reliability and application guarantees [3]. Although our framework enables the reconfiguration of any system mechanism, we focused on QoS on the application-level. MiLAN [6] is a middleware platform, also supporting quality requirements. It is proactive, meaning that it is capable of influencing its mechanisms according to higher level requirements. It aims at utilising only the most suitable set of resources to meet the actual requirements, although its scheme of expressing quality requirements is quite limited. Similarly, MASTAQ [9] is a middleware, supporting quality of service requirements, which uses a dynamic quality management mechanism, similar to our solution.

Existing QoS-aware middleware platforms do not involve the operating system in the reconfiguration process. To the best of our knowledge, there is no resource management framework for sensor networks integrating the same concepts ranging from operating system to middleware, and also enabling quality-aware dynamic reconfiguration.

7 Conclusion and future work

The contribution of this paper is twofold. First, we proposed a system software that integrates a real-time lightweight operating system and a publish/subscribe middleware. The resulted framework provides similar data-centric abstractions for the development of both local and networked applications. It results in easier application development, because shields developers from the possibly distributed nature of the system.

By extending this system support we built a resource management framework that supports quality-aware adaptation. It allows the system to tune quality levels according to application requirements and actual resource availabilities. Its feedback-based control loop mechanism continuously evaluates the resultant quality metrics and satisfies application QoI requirements by tuning knobs of possibly distributed applications.

Future research will look into combining the publish/subscribe middleware with the MAC protocol, because it might result in additional energy savings. Influencing the routing hierarchy based on the distribution of subscribers might result in a more efficient routing tree. The time slot allocation could also be changed based on the amount of information to be disseminated, consequently influencing throughput and delay of the data dissemination.

As the experiments pointed out, designing a closed loop controller is not straightforward even in case of a simplistic controllable quantity. The quality should converge into the desired range rapidly, however transient events should

be suppressed. We aim at developing models to facilitate quality management based on higher level quality concerns. Moreover, we aim at making the quality-management scheme more distributed, applying local control mechanisms.

References

1. I.F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. Wireless sensor networks: A survey. *Computer Networks*, 38:393–422, 2002.
2. Philippe Bonnet, Johannes Gehrke, and Praveen Seshadri. Querying the physical world. *IEEE Personal Communications*, October 2000.
3. Dazhi Chen and Pramod K. Varshney. Qos support in wireless sensor networks: A survey. In *Proc. of the 2004 Int. Conf. on Wireless Networks (ICWN 2004)*, June 2004.
4. Stephen Deering, Deborah L. Estrin, Dino Farinacci, Van Jacobson, Ching-Gung Liu, and Liming Wei. The PIM architecture for wide-area multicast routing. *IEEE/ACM Trans. Netw.*, 4(2):153–162, 1996.
5. Stefan Dulman. *Data-centric architecture for wireless sensor networks.* PhD thesis, Centre for Telematics and Information Technology, Univ. of Twente, The Netherlands, October 2005.
6. Wendi B. Heinzelman, Amy L. Murphy, Hervaldo S. Carvalho, and Mark A. Perillo. Middleware to support sensor network applications. *IEEE Network Magazine*, 18(1):6–14, 2004.
7. Jason Hill, Robert Szewczyk, Alec Woo, Seth Hollar, David Culler, and Kristofer Pister. System architecture directions for networked sensors. In *Proc. of the 9th international conference on Architectural support for programming languages and operating systems*, pages 93–104, Cambridge, Massachusetts, United States, 2000.
8. T. Hofmeijer, S. Dulman, P. G. Jansen, and P. J. M. Havinga. AmbientRT - real time system software support for data centric sensor networks. In *2nd Int. Conf. on Intelligent Sensors, Sensor Networks and Information Processing*, pages 61–66, Melbourne, Australia, Dec 2004. IEEE Computer Society Press.
9. Inseok Hwang, Qi Han, and Archan Misra. MASTAQ: A Middleware Architecture for Sensor Applications with Statistical Quality Constraints. In *The First Int. Workshop on Sensor Networks and Systems for Pervasive Computing (PerSeNS 2005)*, Kauai Island, Hawaii, USA, March 2005.
10. P. G. Jansen, S. J. Mullender, P. J. M. Havinga, and J. Scholten. Lightweight EDF scheduling with deadline inheritance. Technical report TR-CTIT-03-23, Centre for Telematics and Information Technology, Univ. of Twente, The Netherlands, May 2003.
11. Omnet++ discrete event simulator. http://www.omnetpp.org.
12. Anu Purhonen and Esa Tuulari. *Ambient intelligence and the development of embedded system software*, pages 51–67. Kluwer Academic Publishers, Norwell, MA, USA, 2003.
13. Chien-Chung Shen, Chavalit Srisathapornphat, and Chaiporn Jaikaeo. Sensor information networking architecture and applications. *IEEE Personal Communications Magazine*, 8(4):52–59, August 2001.
14. L. van Hoesel and P.Havinga. A lightweight medium access protocol (LMAC) for wireless sensor networks: Reducing preamble transmissions and transceiver state switches. In *Proc. of 1st Int. Workshop on Networked Sensing Systems*, 2004.